ELEN0062 - Introduction to machine learning
First assignment: Feedback

Jean-Michel Begon and Antonio Sutera

University of Liege

November 21, 2018

Outline

Decision tree

k-Nearest neighbors

Linear discriminant analysis

LDA implementation

Presentation and style

Decision tree — Boundaries

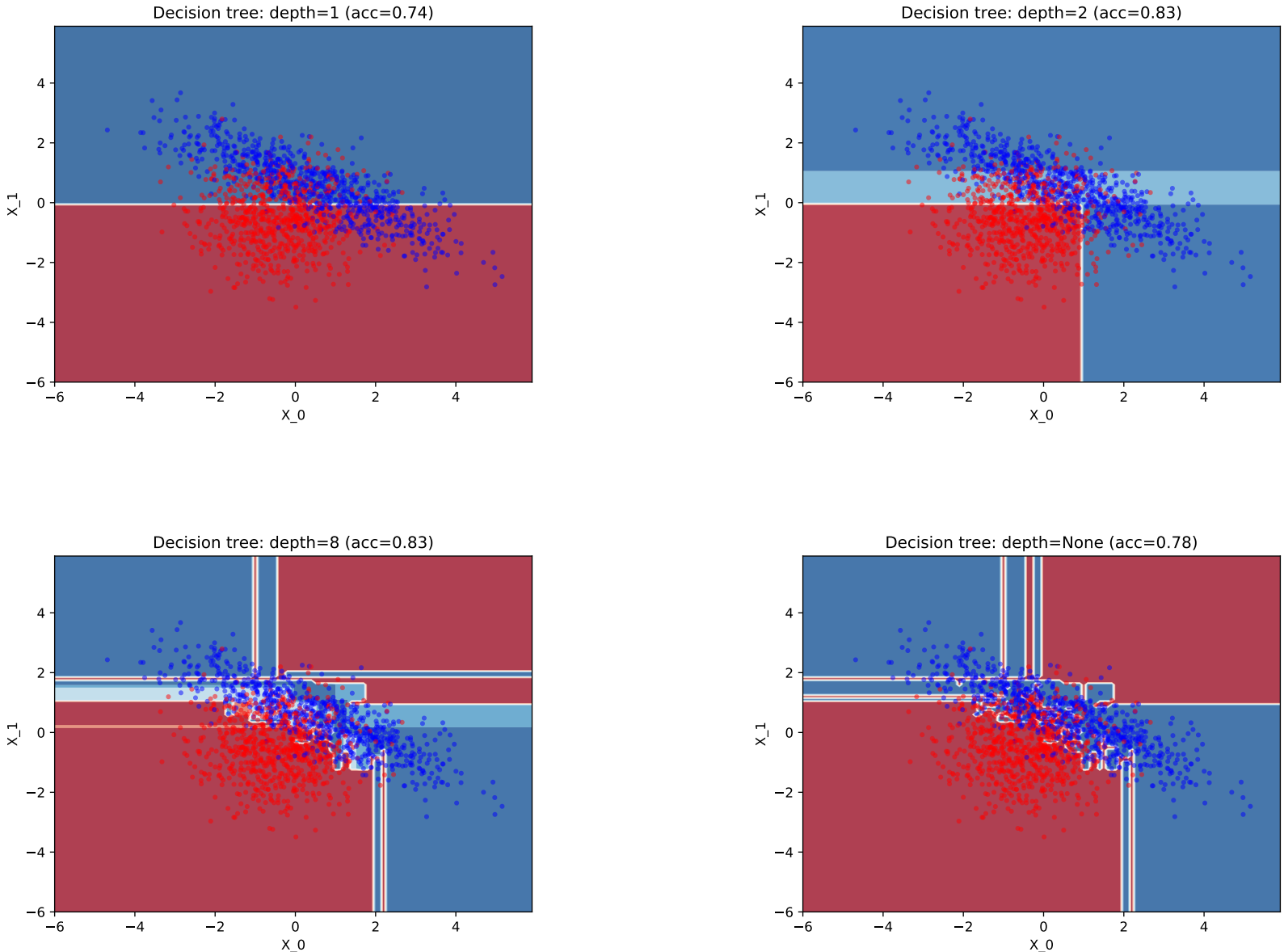

Figure: Decision tree boundary for several depths (dataset 2)

Decision tree — Comments

1. The DT models partition the space with axis-aligned cuts
   - because it splits the dataset by thresholding one of the features.
2. Increasing the depth sub-partition the space (exponentially)
   - because each node of depth $d-1$ is split (unless it is pure).
3. We can see underfitting for depth 1 and 2
   - the boundary is too simple to account for the data.
4. We can see overfitting from depth 8 on
   - too many small regions too specific for the data.
5. The model is confident because the training set is perfectly classified and the model predicts for a zone the proportion of training objects that fell into that zone.

k-Nearest neighbors — Boundaries

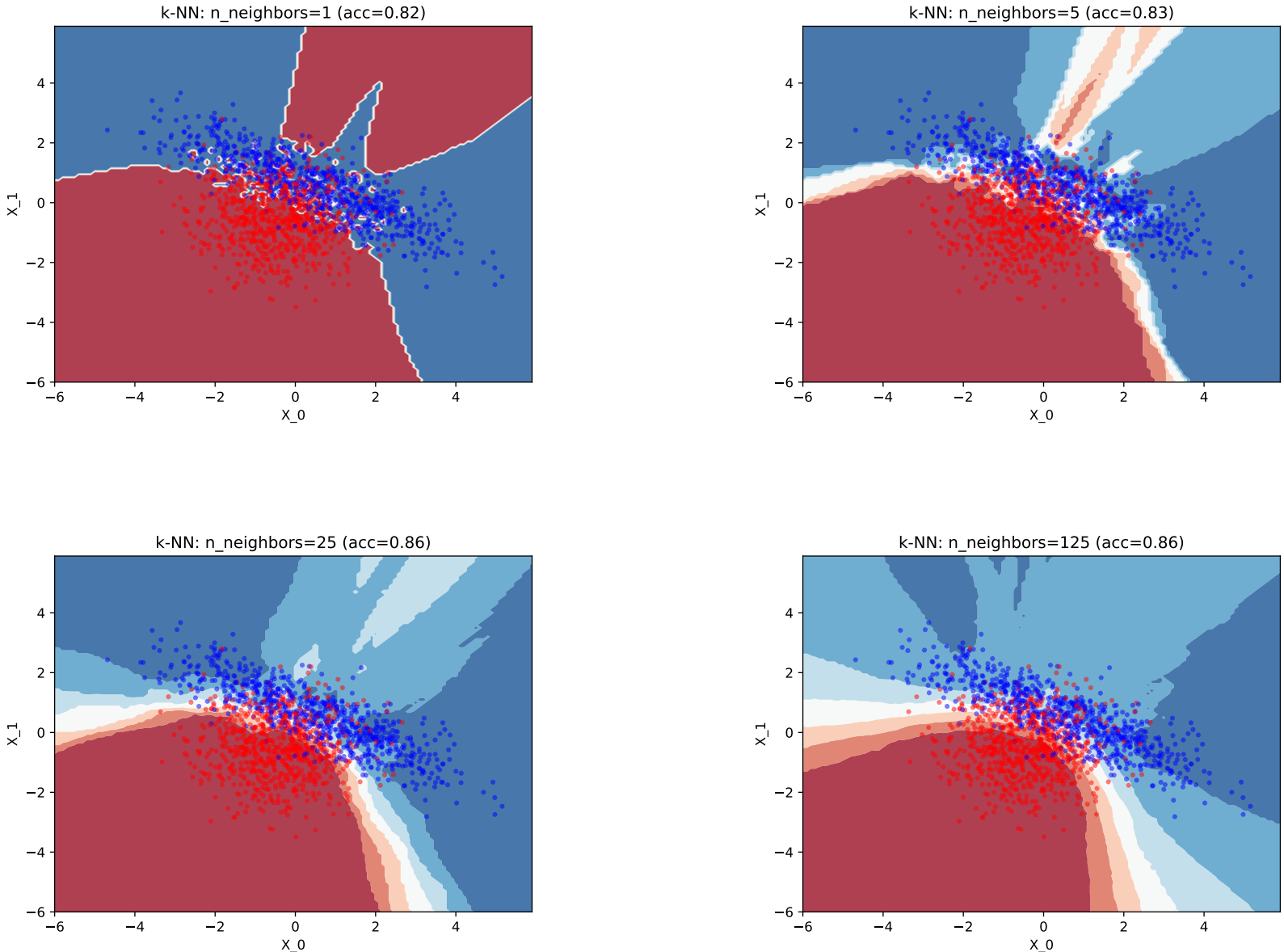

Figure: k-NN boundary for several depths (dataset 2)

k-Nearest neighbors — Boundaries (cont.)

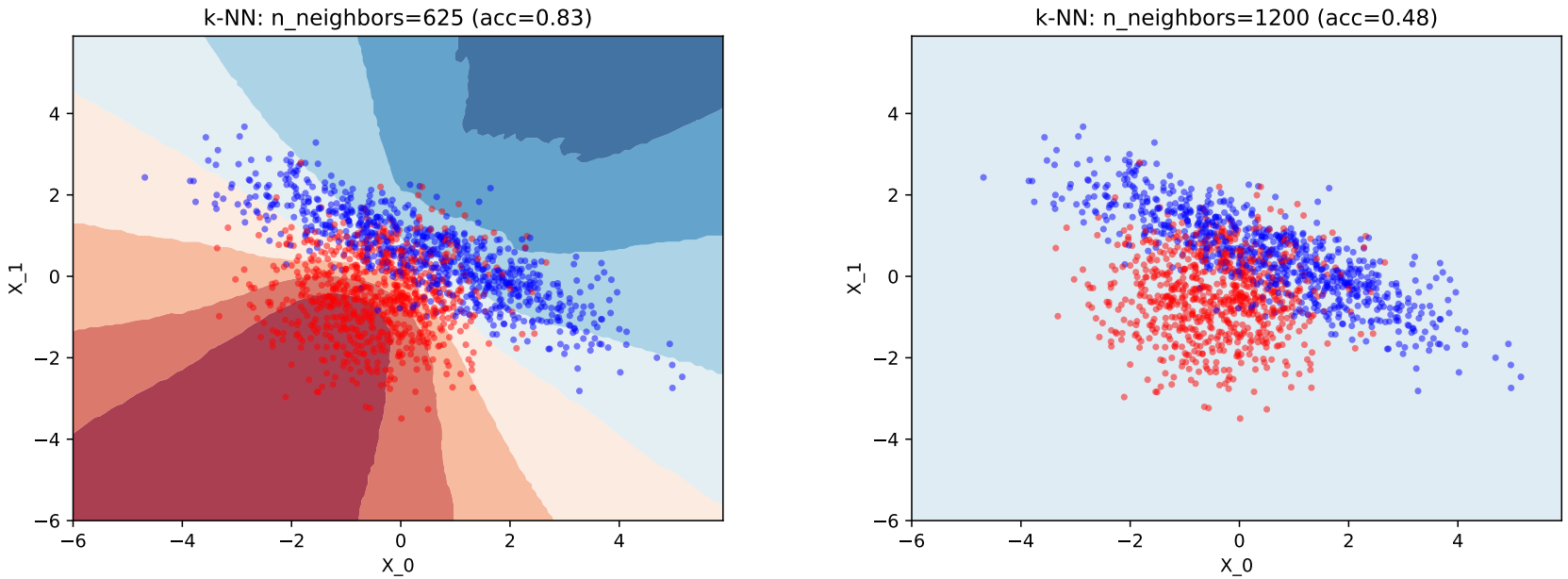

Figure: k-NN boundary for several depths (dataset 2)

k-Nearest neighbors — Comments

The decision for $k = 1$ depicts overfitting: small, arbitrary zones in the middle of the cloud that are only relevant for the training set. On the top part, the red zone seems unatural. The bottom, quadratic, part feels right, though.

We can observe a smoothing of the zones when increasing the number of neighbors and disappearance of the small island. When $k = 5+$, there is no more overfitting: the decision boundary is quite reasonable. We see a quadratic decision boundary at the bottom. Interestingly, there is a zone of less confidence on the top (perpendicular to the ellipse).

By $k = 625$, every point has a neighborhood of half the dataset. Except for points which are on the far off, there is a big ambiguity. Also, the decision boundary is almost linear; we are seeing underfitting.

When $k = 1200$, we see encompass all the training set, wherever we are. Therefore, the prediction is spatially uniform. The actual class that is predicted is only due to the random splitting of the data.

Optimal number should be between $k = 5$ and $k = 125$.

Linear discriminant analysis

Starting from the class density functions:

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)} \tag{1}$$

where $x \in \mathbb{R}^p$ is the feature vector and μ_k and Σ_k are respectively the mean and covariance matrix corresponding to class k.

We need to find the class

$$\arg\max_k P(y = k|x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^K f_l(x)\pi_l} \tag{2}$$

$$= f_k(x)\pi_k \tag{3}$$

where $\pi_k = P(y = k)$ $(k = 1, \ldots, K)$.

---

(2) $\rightarrow$ (3) the denominator is constant wrt to x for all k.

Linear discriminant analysis — Demonstration

$$\gamma_k(x) \triangleq \log\left(f_k(x)\pi_k\right) \tag{4}$$

$$= C_k - \frac{1}{2}\left(x-\mu_k\right)^T \Sigma_k^{-1}\left(x-\mu_k\right) + \log \pi_k \tag{5}$$

$$= C_k - \frac{1}{2}x^T\Sigma_k^{-1}x + \frac{1}{2}\mu_k^T\Sigma_k^{-1}x + \frac{1}{2}x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k + \log \pi_k \tag{6}$$

$$= D_k(x) + \frac{1}{2}\left(\mu_k^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k\right) - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k + \log \pi_k \tag{7}$$

$$= D_k(x) + \mu_k^T\Sigma_k^{-1}x - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k + \log \pi_k \tag{8}$$

---

(4) → (5) by definition of $f_k(x)$.

(5) → (6) distribution.

(6) → (7) posing $D_k(x) = C_k - \frac{\mathbf{1}}{\mathbf{2}}x^T\Sigma_k^{-\mathbf{1}}x$.

(7) → (8) $\mu_k^T\Sigma_k^{-\mathbf{1}}x = x^T\Sigma_k^{-\mathbf{1}}\mu_k$ since $\Sigma_k^{-\mathbf{1}}$ is symmetric (since Σ_k is symmetric).

Linear discriminant analysis — Demonstration (cont.)

By homoscedasticity:

$$\Sigma_k^{-1} = \Sigma_l^{-1} = \Sigma^{-1}, \quad \forall k, l \tag{9}$$

$$C_k = C_l = C, \quad \forall k, l \tag{10}$$

$$D_k(x) = D_l(x) = D(x) = C + -\frac{1}{2} x^T \Sigma^{-1} x, \quad \forall k, l \tag{11}$$

$$\gamma_k(x) = D(x) + \mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \tag{12}$$

Going back to the prediction of the class

$$\arg\max_k \gamma_k(x) = \arg\max_k \left(D(x) + \mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \right) \tag{13}$$

$$= \arg\max_k \left(\mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \right) \tag{14}$$

$$= \arg\max_k \delta_k(x) \tag{15}$$

---

(13) → (14) since $D(x)$ does not depend on the class and is content wrt x.

Linear discriminant analysis — Demonstration (cont.)

$\delta_k(x)$ is called the linear discriminant function of class k and is linear:

$$\delta_k(x) = \mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k = \alpha_k^T x + \beta_k \tag{16}$$

In the binary case, the classification rule becomes

$$\begin{cases} \text{class } k, & \text{if } \delta_k(x) > \delta_l(x) \iff (\alpha_k^T - \alpha_l^T)x + (\beta_k - \beta_l) > 0 \\ \text{class } l, & \text{otherwise} \end{cases} \tag{17}$$

Binary linear.

Multi-class piecewise linear.

Linear discriminant analysis — Results

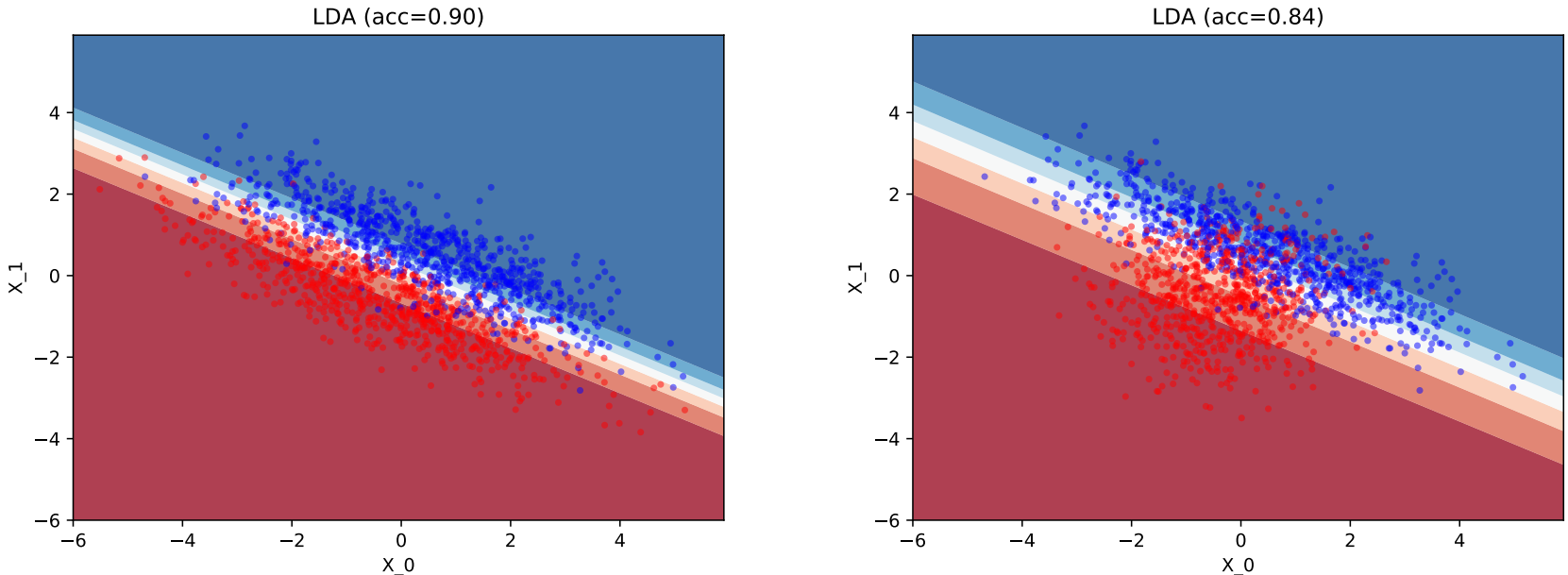

Figure: LDA boundary for several depths (dataset 2)

Accuracy	LDA	kNN ($k = 125$)
Dataset 1	0.91 ± 0.01	0.91 ± 0.01
Dataset 2	0.85 ± 0.03	0.85 ± 0.01

Table: Accuracies for LDA and kNN

Linear discriminant analysis — Discussion

The discussion must encompass both optimality and accuracy.

In the case of dataset 1, LDA is optimal (basically same demonstration).

In the case of dataset 2, the optimal decision boundary is quadratic (cf. k-NN), however, the linear decision is working surprisingly well; it is not because the assumptions are not met that the model is bad.

The difference in accuracies is not so much because of the homoscedasticity, but rather because of the shape of the data (bayes rate is lower in the second dataset).

LDA implementation

Penalty only if the code was not genuine or not generic

- Use of Scikit-learn implementation
- Only works for two input features
- Only works for binary classifications

But implementations could be improved:

- Efficiency
- Numerical stability

LDA implementation — Efficiency

$$\delta_k(x) = \mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k = \alpha_k^T x + \beta_k \tag{20}$$

- α_k and β_k can only be computed once (at training time).
- β_k can be computed from α_k:

$$\beta_k = -\frac{1}{2} \alpha_k^T \mu_k + \log \pi_k \tag{18}$$

- Everything can be stored together (avoid loops by using NumPy vectorized operations):

$$\delta(x) = \begin{bmatrix} \delta_1(x) \\ \vdots \\ \delta_K(x) \end{bmatrix} = \begin{bmatrix} \alpha_1^T \\ \vdots \\ \alpha_K^T \end{bmatrix} x + \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_K \end{bmatrix} = A^T x + b \tag{19}$$

LDA implementation — Efficiency (cont.)

- There is no need to compute the actual posterior probabilities to predict the class.
- There is no need to compute the actual posterior probabilities:

$$\frac{e^{\delta_k(x)}}{\sum_{l=1}^{K} e^{\delta_l(x)}} = \frac{e^{\gamma_k(x)-D(x)}}{\sum_{l=1}^{K} e^{\gamma_l(x)-D(x)}} \tag{21}$$

$$= \frac{e^{D(x)}}{e^{D(x)}} \frac{e^{\gamma_k(x)}}{\sum_{l=1}^{K} e^{\gamma_l(x)}} \tag{22}$$

$$= \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} f_l(x)\pi_l} = P(y = k|x) \tag{23}$$

LDA implementation — Numerical stability

$$\left(\mu_k^T \Sigma^{-1}\right)^T = \Sigma^{-1}\mu_k = \alpha_k \tag{24}$$

$$\iff \Sigma\alpha_k = \mu_k \tag{25}$$

Or more generally

$$\Sigma A = M \tag{26}$$

We do not actually care for Σ^{-1}, we only need A.

Solving the system $\Sigma A = M$ directly is more stable numerically than inverting the covariance matrix and multiplying by the mean vector.

LDA implementation — Putting it all together

1. Assess parameters P, M and Σ from data
2. Compute A by solving $\Sigma A = M$
3. Compute b from A, M and P

Σ is the covariance matrix of size $p \times p$.

$$P = \begin{bmatrix} \log \pi_1 \\ \vdots \\ \log \pi_K \end{bmatrix}$$

is the log prior vector of size $K \times 1$.

$$M = \begin{bmatrix} \mu_1^T \dots \mu_K^T \end{bmatrix}$$

is the concatenation of the means vector of size $p \times K$.

$$A = \begin{bmatrix} \alpha_1^T \dots \alpha_K^T \end{bmatrix}$$

is the coefficient matrix of size $p \times K$.

$$b = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_K \end{bmatrix}$$

is the intercept vector of size $p \times 1$.

LDA implementation — Putting it all together

Prediction

given

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

compute

$$\hat{Y} = \begin{bmatrix} \delta(x_1)^T \\ \vdots \\ \delta(x_n)^T \end{bmatrix} = XA \oplus b^T$$

where $P \oplus d$ adds the row vector d to each row of matrix P.

Either take the maximum rowwise to get the class or normalize to get the probabilities.

LDA implementation — Estimating the parameters

```
def fit(self, X, y):
    ...
    uniques = np.unique(y)
    n_classes = len(uniques)
    n_samples, n_features = X.shape

    ### Estimating parameters from the data
    # Prior: [n_classes]
    priors = np.histogram(y, n_classes)[0] / n_samples
    # M [n_features, n_classes]
    means = np.zeros((n_classes, n_features))
    X_centered = X.copy()
    for i in range(n_classes):
        idx = y == i
        means[:, i] = np.mean(X[idx], axis=0)
        X_centered[idx] -= means[i]
```

LDA implementation — Estimating the model parameters

```
    # Sigma [n_features, n_features]
    df = n_samples - n_classes
    covariance = np.dot(X_centered.T, X_centered) / df

    ### Computing the coefficients
    # Coefficients: A [n_features, n_classes]
    self.coef_ = np.linalg.solve(covariance, means)
    # Intercept: b [n_classes]
    self.intercept_ = -0.5 * np.sum(self.coef_.T*means,
                                    axis=1)
                      + np.log(priors)

    return self
```

LDA implementation — Prediction

```
def predict_raw(self, X):
    return np.dot(X, self.coef_) + self.intercept_


def predict(self, X):
    return np.argmax(self.predict_raw(X), axis=1)


def predict_proba(self, X):
    exp = np.exp(self.predict_raw(X))
    for i in range(exp.shape[1]):
        exp[:, i] = exp[:, i] / exp.sum(axis=1)
    return exp
```

Presentation and style

Of content and style

Sometimes in academia:

$$\text{grade} = \frac{\text{content} + \text{style}}{2}$$

Pay attention to both!

Everywhere else:

$$\text{grade} = \sqrt{\text{content} \times \text{style}}$$

A few advices

- Do not put figures in appendix if you describe them in the text (same goes for demonstration).
- Analyze is more than describe: show you have understood (make connections with the relevant part of the theory).
- Make a proper bibliography and cite your sources.
- Do not copy paste: rephrase to show you have understood.
- Choose carefully how you represent data.